ILGC SEMESTER 7 MID SEM PRESENT

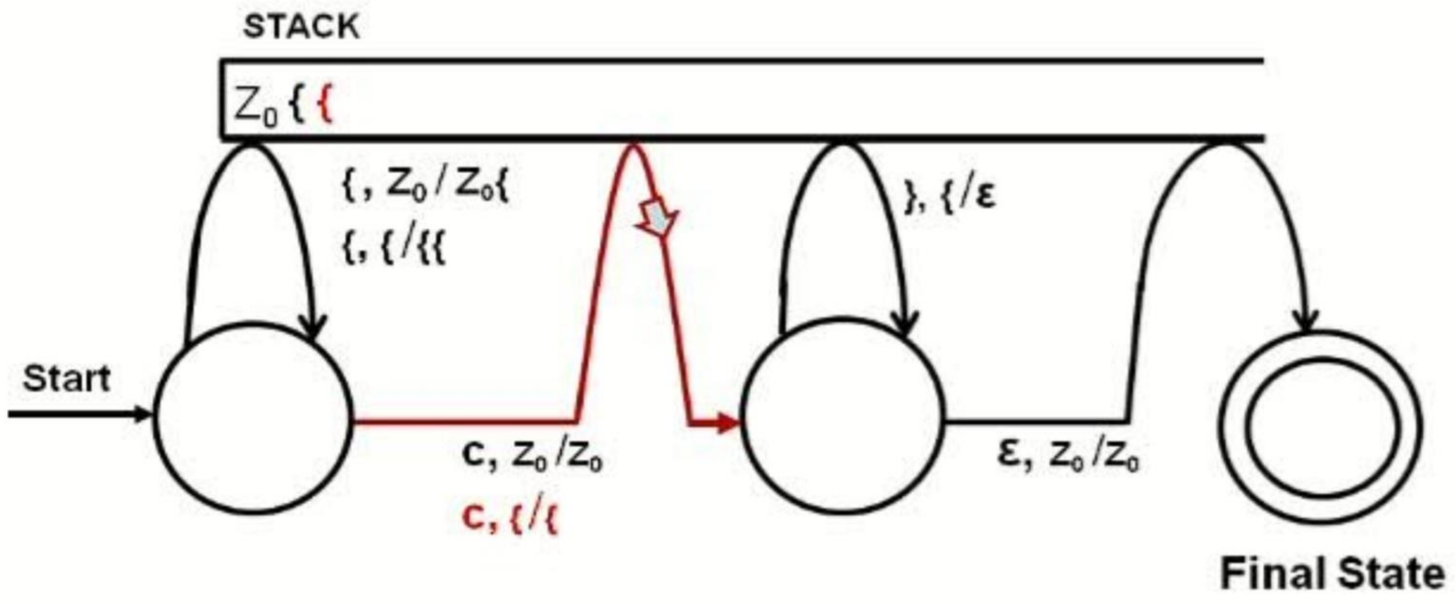
Dhruv Menon

WHAT MAKES THIS DIFFERENT FROM EXISTING ONES?

- Existing ones just take the transition function of the PDA as input and tell you whether it accepts or rejects the input. However, the user is not able to visualize how the machine makes a decision on an input string. They are not able to visualize an animation of the stack,s , tape and the symbols(stack alphabet, input symbol, push/pop operation) etc. The condition for whether the input a context free language is defined to be when the input ends ,the final state of the machine is one of the accept states and the stack is empty i.e the stack alphabet is "*".

- Show example to Professor

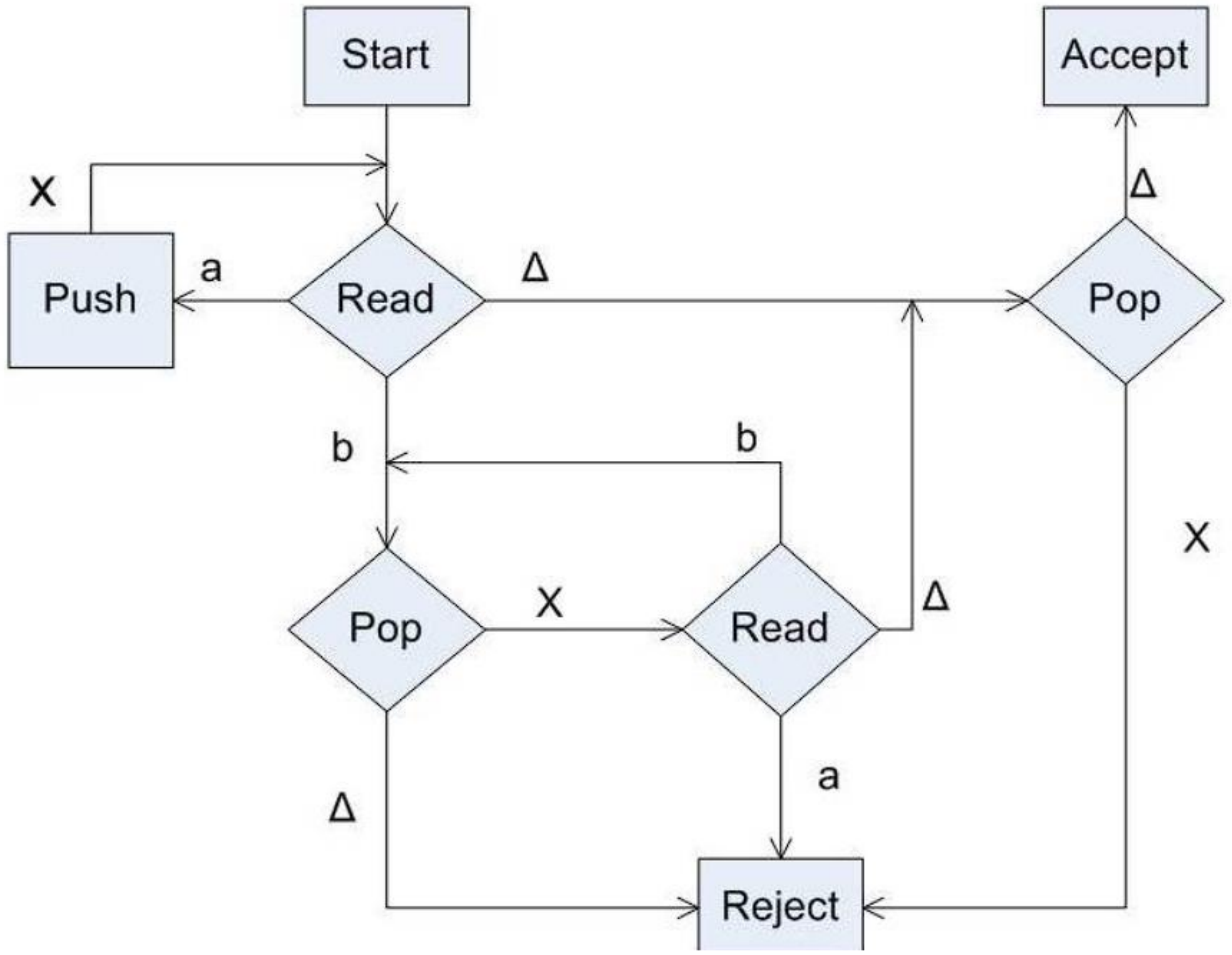
READ ONLY INPUT BUFFER: {{c}}



PUSH DOWN AUTOMATON DESIGN

- Pushdown Automata (PDA) are acceptors of context-free languages which include programming languages such as C++ and Java. A Pushdown Automaton for a non-regular Context-Free Language, L_m = **{** $\{^n c\}^n$: where n >=0 }, is shown in action with an input string **"{{c}}"** . L_m is a language with matching number of {'s and }'s seperated by one "c". Programming languages such as Java have balanced {'s and }'s. The machine starts execution from the start state. The moving arrow shows the execution path and the underlined input symbol shows the current symbol being processed. Zo is the start symbol which is used as the bottom marker of the stack following Hopcroft, Motwani, & Ullman, (2007) *Introduction to Automata Theory, Languages, and Computation*

VISUALIZATION



TECHNICAL DETAILS(HOPCROFT)

- Definition of Pushdown Automata
  A pushdown automaton (PDA) is a collection of eight things:
  ï¿½ An alphabet ∑ of input letters.
  ï¿½ An input TAPE (infinite in one direction), which initially contains the input string to be processed followed by an infinite number of blanks, Δ
  ï¿½ An alphabet E of STACK characters.
  ï¿½ A pushdown STACK (infinite in one direction), which initially contains all blanks, Δ.
  ï¿½ One START state that has only out-edges, no in-edges. Can have more than one arc leaving the START state. There are no labels on arcs leaving the START state.
  ï¿½ Halt states of two kinds:
  1. zero or more ACCEPT states
  2. zero or more REJECT states
  □ Each of the Halt states have in-edges but no out-edges.
  □ Finitely many nonbranching PUSH states that introduce characters from E onto the top of the STACK.
  □ Finitely many branching states of two kinds:
  1. READ states, which read the next unused letter from TAPE and may have out-edges labeled with letters from ∑ or a blank, Δ. (There is no restriction on duplication of labels and no requirement that there be a label for each letter of ∑, or Δ.)
  2. POP states, which read the top character of STACK and may have out-edges labeled with letters of E and the blank character Δ, with no restrictions.

ANIMATION DETAILS

- **1. Animation Design**
- **Path Coloring**: Each NPDA path (due to nondeterminism) assigned a unique RGB color (0-100% each → ~1M colors).
- **Contrast Optimization**: Ensure high contrast between adjacent paths via DFS traversal; switch color on branch termination.
- **Dynamic Palette**: Initialize color count based on input length (`l`), increment as paths diverge.
- **2. User Interface Features**
- **Drag-and-Drop Builder**:
  - **States** as circles, **Stack** as a rectangle (drag stack alphabets into it).
  - **Transitions** as arrows between states (drag input alphabets onto arrows).
- **Workflow**:
  - **Confirm Button** to finalize design → start animation.
  - **Input String Entry**: Execute NPDA on user-provided input.
  - **Navigation**: Use → (next step) and ← (previous step) keys.
  - **3. Visualization Elements**
- **Real-Time Stack Animation**: Dynamically display stack operations (push/pop).
- **Highlighting**:
  - **Current Tape Cell**: Bold highlight for active input position.
  - **Top of Stack**: Emphasize the top stack alphabet.

- **Key Takeaway**: Combines intuitive design with advanced visualization for NPDA path exploration and debugging.

ALTERNATIVE ANIMATION BASED ON COHEN 2007

- **1. PDA Definition (Cohen 2007)**
- **8 Components**:
  - Input alphabet (∑), infinite TAPE (input + Δ), STACK alphabet (E), infinite STACK (initially Δ).
  - **States**:
    - **START** (no in-edges, unlabeled out-edges).
    - **HALT**: ACCEPT/REJECT (no out-edges).
    - **Nonbranching PUSH** (add stack characters).
    - **Branching**: READ (process TAPE input/Δ) and POP (process STACK top/Δ).

    2. Simplified Animation Design
- **Graphical States**: Represented as labeled rectangles (Start, Accept, Push, Read, Pop).
- **Transitions**:
  - **READ → POP**: Labeled arrows based on TAPE input.
  - **POP → New State**: Arrows labeled with popped STACK character (Δ = empty stack; acceptance requires Δ).
  - **PUSH**: Arrows indicate stack character added.
- **Stack Visualization**:
  - Bold highlight on top character.
  - Popping Δ triggers stack-empty check for acceptance.
  - **3. Technical Implementation**
- **Framework**: JavaScript/HTML for dynamic rendering.
- **Fluid Animation**: High FPS for smooth transitions.
- **Key Visuals**:
  - Directed graph with labeled states/transitions.
  - Real-time TAPE/STACK updates.

  - **Key Advantage**: Streamlined visualization focusing on core PDA operations (READ, POP, PUSH) for clarity and educational use.

APPLICATIONS

- Check if strings satisfying a specific pattern are not context-free
- Teaching Theory of Computation

REFERENCES

- [1] Hopcroft, J., Motwani, R., and Ullman, J. Introduction to Automata Theory, Languages, and Computation, Pearson Education, 2007
  [2] Cohen, D., Introduction to Computer Theory, 2nd Edition, New York, NY: John Wiley & Sons, 1997.
  [3] Lewis, H. R., and Papadimitriou, C. H., Elements of the Theory of Computation, 2nd Edition, New Jersey: Prentice_Hall, 1998.
  [4] W. A. Woods, Transition network grammars for natural language analysis, Communications of the ACM, v.13 n.10, p.591-606, Oct. 1970
  [5] Sipser, M., Introduction to the Theory of Computation, 2nd Edition, PWS Publishing, 2006.
  [6] Goddard, W. (2009) Introducing the Theory of Computation, Jones & Bartlett Publishers.
  [7] Kozen, D. (2006). Theory of Computation. Springer . [
- 8] Rich, E. (2007) Automata, Computability and Complexity: Theory and Applications